

Fonctions réseau du noyau Linux

Philippe Latu

philippe.latu(at)inetdoc.net

Laurent Foucher

laurent.foucher(at)iut-tlse3.fr

<http://www.inetdoc.net>

Le catalogue des fonctions réseau du noyau Linux étant assez conséquent, cette introduction n'a pas pour but d'être exhaustive. Dans un premier temps, on se propose d'identifier le sous-système réseau dans l'architecture du noyau Linux. Dans un second temps, on présente l'utilisation de quelques fonctions réseau caractéristiques. L'objectif est de fournir un panorama général de l'utilisation des nombreuses fonctions réseau fournies avec le noyau Linux.

Table des matières

1. Copyright et Licence	1
1.1. Méta-information	2
1.2. Conventions typographiques	2
2. Présentation du noyau LINUX	2
2.1. Introduction	2
2.2. Architecture du système GNU/Linux	2
3. Sous-système réseau du noyau LINUX	4
3.1. Packet Socket	4
3.2. Kernel/User netlink socket	5
3.3. Socket Filtering	5
3.4. Unix domain socket	5
3.5. TCP/IP networking	6
3.5.1. IP: multicasting	6
3.5.2. IP: advanced router	6
3.5.3. IP: kernel level autoconfiguration	7
3.5.4. IP: optimize as router not host	7
3.5.5. IP: tunneling	7
3.5.6. IP: GRE tunnel over IP	7
3.5.7. IP: TCP Explicit Congestion Notification support	7
3.5.8. IP: TCP syncookie support	7
3.5.9. IP: Allow large windows (not recommended if <16 Mb of memory)	8
3.5.10. Network packet filtering (replace ipchains)	8
3.6. 802.1Q VLAN Support	12
3.7. The IPX Protocol	13
3.8. Appletalk DDP	13
3.9. DECnet support	13
3.10. 802.1d Ethernet Bridging	13
4. Les outils réseaux du noyau Linux	13
4.1. Configuration des interfaces réseaux	13
4.2. ip link	13
4.3. ip address	13
4.4. ip rule	14
4.5. ip route	14
5. Configuration du filtrage	15
5.1. Introduction	15
5.2. Netfilter et iptables	15
5.3. Utilisation de l'outil iptables	15
5.4. Le filtrage avec iptables	16
5.5. Le suivi des communications, <i>stateful firewalling</i>	16
5.6. La traduction d'adresse (NAT)	17

1. Copyright et Licence

Copyright (c) 2000,2012 Philippe Latu, Laurent Foucher.
Permission is granted to copy, distribute and/or modify this document under the terms of the GNU Free Documentation License, Version 1.3 or any later version published by the Free Software Foundation; with no Invariant Sections, no Front-Cover Texts, and no Back-Cover Texts. A copy of the license is included in the section entitled "GNU Free Documentation License".

Copyright (c) 2000,2012 Philippe Latu, Laurent Foucher.
Permission est accordée de copier, distribuer et/ou modifier ce document selon les termes de la Licence de Documentation Libre GNU (GNU Free Documentation License), version 1.3 ou toute version ultérieure publiée par la Free Software

Foundation ; sans Sections Invariables ; sans Texte de Première de Couverture, et sans Texte de Quatrième de Couverture. Une copie de la présente Licence est incluse dans la section intitulée « Licence de Documentation Libre GNU ».

1.1. Méta-information

Cet article est écrit avec *DocBook*¹ XML sur un système *Debian GNU/Linux*². Il est disponible en version imprimable au format PDF : [linux.networking.pdf](#)³.

1.2. Conventions typographiques

Tous les exemples d'exécution des commandes sont précédés d'une invite utilisateur ou *prompt* spécifique au niveau des droits utilisateurs nécessaires sur le système.

- Toute commande précédée de l'invite \$ ne nécessite aucun privilège particulier et peut être utilisée au niveau utilisateur simple.
- Toute commande précédée de l'invite # nécessite les privilèges du super-utilisateur.

2. Présentation du noyau LINUX

2.1. Introduction

Linux s'intègre dans la longue histoire des systèmes UNIX. Le développement de ce système d'exploitation a débuté en 1969 sous l'impulsion de Ken Thompson et Dennis Ritchie qui travaillaient alors pour la société Bell Laboratories. Plusieurs versions furent développées en interne et c'est en 1975 qu'apparut la version 6 qui deviendra la base des Unix commerciaux.

Par la suite, de nombreuses implémentations d'UNIX furent développées. L'université de Berkeley fut à la base de la version BSD, Hewlett Packard proposa la version HP-UX, etc... Malgré de bonnes intentions au départ, il existait des incompatibilités entre tous ces Unix, si bien que le portage d'une application d'un UNIX vers un autre était difficile. Pour réduire ces disparités, la société AT&T proposa un standard UNIX en 1983, connu sous le nom de System V. En 1986, l'Institute of Electrical and Electronics Engineers (IEEE) proposa un autre standard connu sous le terme de POSIX. POSIX est une standardisation permettant d'assurer la portabilité des applications d'un UNIX à un autre.

Le système d'exploitation GNU/Linux se comporte comme un UNIX et implémente les spécifications POSIX, avec des extensions système V et BSD.

Issu du travail d'un étudiant finlandais, Linus Torvalds, Linux se distingue par le fait qu'il est distribué sous les conditions d'une licence particulière, appelée GPL (GNU Public License). Cette licence précise que toute personne peut modifier, améliorer ou corriger le code source, mais que ces modifications devront également être distribuées librement.

Les principales caractéristiques de Linux sont les suivantes :

- Multitâches : exécute plusieurs programmes en même temps.
- Multi-utilisateurs : plusieurs utilisateurs peuvent être actifs en même temps.
- Multi plates-formes : Linux peut fonctionner avec différents types de processeurs (Intel, Sparc, Alpha, PowerPC, etc...).
- Supporte un grand nombre de systèmes de fichiers : Ext(2|3|4), XFS, FAT, VFAT, NFS, CIFS, etc.
- Dispose d'un catalogue de fonctions réseau conséquent. Voir [Section 3, « Sous-système réseau du noyau LINUX »](#).

L'évolution du noyau est très rapide et celui qui est utilisé pour ce programme de formation appartient à la série 2.6.

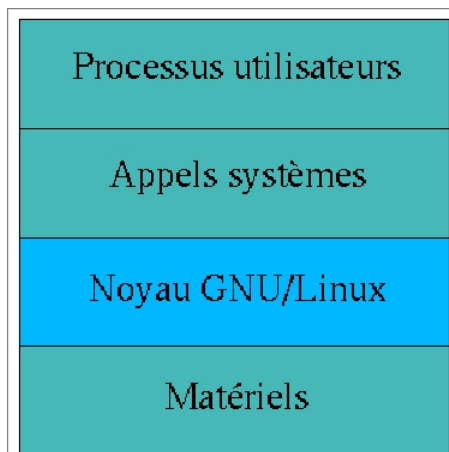
2.2. Architecture du système GNU/Linux

Comme dans tout système d'exploitation, le noyau LINUX est une interface entre des programmes et des périphériques physiques. L'accès à ces périphériques se fait par l'intermédiaire d'appels systèmes qui sont identiques quelle que soit la machine. Cette encapsulation du matériel libère les développeurs de logiciels de la gestion complexe des périphériques : c'est le système d'exploitation qui s'en charge. Ainsi, si le système d'exploitation existe sur plusieurs architectures, l'interface d'utilisation et de programmation sera la même sur toutes. On dira alors que le système d'exploitation offre une *machine virtuelle* à l'utilisateur et aux programmes qu'il exécute.

¹ <http://www.docbook.org>

² <http://www.debian.org>

³ <http://www.inetdoc.net/pdf/linux.networking.pdf>



GNU/Linux est considéré comme un système d'exploitation monolithique, écrit comme un ensemble de procédures qui peuvent s'appeler mutuellement. Pour l'utilisateur, il se présente comme un seul gros fichier. Cependant, il contient un ensemble de composants réalisant chacun une tâche bien précise. Cette construction monolithique induit un aspect important : la notion d'*espace noyau (kernel space)* et d'*espace utilisateur (userspace)*. Dans l'espace noyau, aucune restriction n'est imposée. Dans l'espace utilisateur, un certain nombre de restrictions sont imposées (par exemple, la création d'un fichier ne peut se réaliser que si les droits sont suffisants), et le processus ne peut avoir accès qu'aux zones mémoires qui lui ont été allouées.

Le noyau LINUX est composé de cinq sous-systèmes principaux. Un sous-système peut être défini comme une entité logicielle qui fournit une fonctionnalité particulière.

Tâches réalisées par le noyau LINUX

Gestion des processus, Ordonnanceur, *Scheduler*

Ce sous-système est chargé de répartir équitablement les accès au processeur entre toutes les applications actives. Cela n'inclut pas seulement les processus utilisateurs, mais aussi les sous-systèmes du noyau lui-même. Cette fonction est réalisée par le *Scheduler*.

Gestion de la mémoire

Ce sous-système est chargé d'affecter à chaque programme une zone mémoire. Il a également un rôle de protection : la mémoire pour un processus est privée et celle-ci ne doit pas être lue ni modifiée par un autre.

Système de fichier virtuel

Le sous-système de fichiers garantit une gestion correcte des fichiers et un contrôle des droits d'accès. Pour limiter la complexité liée aux nombreux systèmes de fichiers existants, LINUX adopte le concept de *Virtual FileSystem (VFS)*. Le principe du VFS est de proposer des appels systèmes identiques quel que soit le système de fichiers. Il est de la responsabilité du noyau de détourner les appels standards vers les appels spécifiques au système de fichiers.

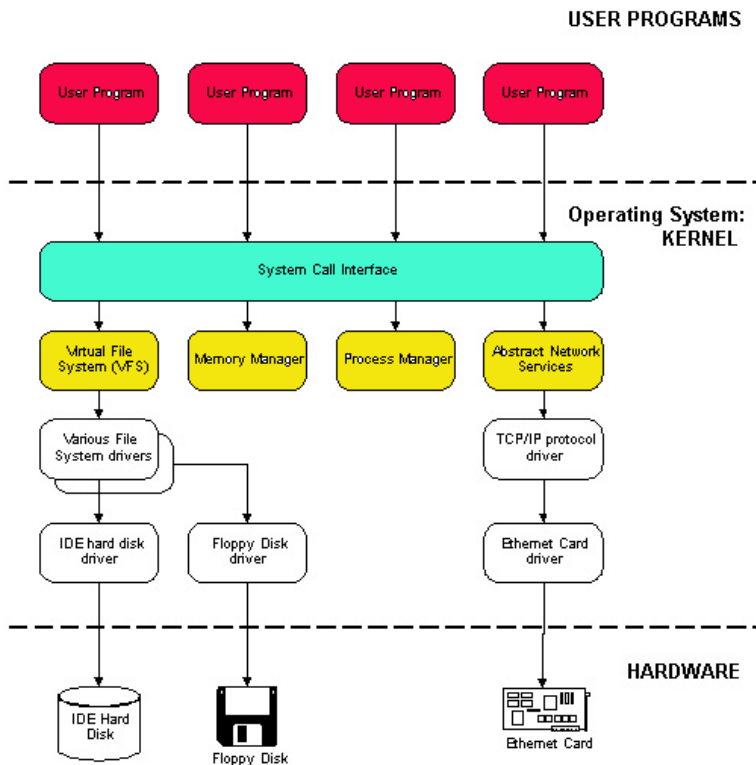
Service réseau

Le sous-système réseau permet à Linux de se connecter à d'autres systèmes à travers un réseau informatique. Il y a de nombreux périphériques matériels qui sont supportés et plusieurs protocoles réseaux peuvent être utilisés.

Communications Inter Processus

Dans la mesure où un processus ne peut avoir accès qu'à la zone mémoire qui lui a été allouée, LINUX propose plusieurs mécanismes permettant à des applications de communiquer entre elles.

Les relations entre les différentes parties du noyau sont montrées sur la figure ci-dessous.



3. Sous-système réseau du noyau LINUX

Networking options

Le but de cette section est de découvrir les diverses options réseau que propose le noyau LINUX.

Les fonctions réseau indépendantes du matériel (piles de protocoles, liste de filtres, etc.) sont regroupées dans les menus Networking puis Networking Options.

Les pilotes de périphériques réseau sont accessibles à partir des menus Device Drivers puis Network device support.

3.1. Packet Socket

Cette fonctionnalité est utilisée pour recevoir ou envoyer des paquets bruts sur les périphériques réseaux sans passer par l'intermédiaire d'un protocole réseau implémenté dans le noyau. Certains programmes, tel que tcpdump, utilisent cette option.

Le terme «socket» désigne l'interface de programmation à travers laquelle l'on va pouvoir accéder aux ressources réseau du noyau. La création d'une interface «socket» est réalisée par l'appel système suivant :

```
int socket(famille, type, protocole);
```

```
int famille;
int type;
int protocole;
```

Le paramètre «famille» permet de préciser avec quel protocole réseau on souhaite travailler. L'ensemble des familles disponibles est listé dans le fichier `/usr/include/linux/socket.h`. Les types définissent le protocole de transport (TCP ou UDP). Une nouvelle famille de socket associée à cette fonctionnalité est ainsi disponible, à savoir `AF_PACKET`.

Sous-option de Packet Socket

mapped IO

Cette option permet d'utiliser un mécanisme d'entrée-sortie plus rapide.

Exemple 1. Utilisation de la socket packet

```
#include <stdio.h>
#include <sys/socket.h>
#include <sys/ioctl.h>
#include <net/if.h>
#include <linux/if_ether.h>
#include <linux/if_packet.h>

main ()
{
```

```

int sock_fd ;
struct sockaddr_ll sll;
struct ifreq ifr;
char buffer[2000];
int nb_octet;

if (sock_fd = socket(AF_PACKET,SOCK_RAW,htons(ETH_P_ALL)) == -1 ) {
    printf("Erreur dans la création de la socket\n");
    return -1 ;
}

memset(&ifr, 0, sizeof(ifr));
strncpy (ifr.ifr_name, "eth0", sizeof(ifr.ifr_name));

if (ioctl(sock_fd,SIOCGIFINDEX, &ifr) == -1 ) {
    printf("Erreur dans la recherche de index\n");
    return -1 ;
}

memset(&sll, 0, sizeof(sll));
sll.sll_family = AF_PACKET ;
sll.sll_ifindex = ifr.ifr_ifindex ;
sll.sll_protocol = htons(ETH_P_ALL);

if (bind(sock_fd, (struct sockaddr *) &sll, sizeof(sll)) == -1) {
    printf("Erreur avec bind\n");
    return -1 ;
};

nb_octet=recvfrom(sock_fd,buffer,sizeof(buffer),0,NULL,0);
printf("Nombre d'octets reçus : %d\n",nb_octet);
}

```

3.2. Kernel/User netlink socket

Kernel/User netlink socket : Définit une nouvelle famille de socket, AF_NETLINK. Cette socket permet d'établir une communication bidirectionnelle entre le noyau et l'espace utilisateur. Cette option est nécessaire pour pouvoir utiliser l'outil `iproute2` qui permet la configuration de la partie réseau du noyau.

En plus de cette socket, la communication peut également se réaliser, pour un processus utilisateur, par la lecture ou l'écriture de fichiers caractères spéciaux. Ces fichiers spéciaux ont le numéro majeur 36 et se trouvent dans le répertoire `/dev`.

Sous-option de netlink

Routing Messages

Le noyau fournit des informations sur le routage via le fichier `/dev/route` de numéro majeur 36 et de numéro mineur 0.

Netlink Device Emulation

Permet la compatibilité avec d'anciennes fonctionnalités. Option amenée à disparaître.

3.3. Socket Filtering

Cette fonctionnalité permet, dans les programmes en mode utilisateur, la mise en place de filtres au niveau des sockets. On a ainsi la possibilité d'autoriser ou d'interdire des types de données traversant une socket. Cette fonctionnalité est dérivée du filtrage de paquets Berkeley. Pour plus d'informations, voir le fichier `Documentation/networking/filter.txt` dans les sources du noyau.

3.4. Unix domain socket

Permet la prise en charge des sockets du domaine UNIX. X-windows et syslog sont des exemples de programmes qui utilisent ce type de fonctionnalité. Les sockets UNIX ne permettent que des communications locales sur une machine. Ce type de socket est lié à la création d'un fichier. Le nom de la famille associé aux sockets du domaine Unix est AF_UNIX.

Exemple 2. Utilisation de la socket UNIX

```

#include <stdio.h>
#include <stdlib.h>
#include <errno.h>
#include <string.h>
#include <sys/types.h>
#include <sys/socket.h>
#include <sys/un.h>

int main(void)
{
    int socket_unix, len;
    struct sockaddr_un local;

    if ((socket_unix = socket(AF_UNIX, SOCK_STREAM, 0)) == -1) {

```

```

    perror("socket");
    exit(1);
}

local.sun_family = AF_UNIX;
strcpy(local.sun_path, "/tmp/test_socket_unix");
unlink(local.sun_path);
len = strlen(local.sun_path) + sizeof(local.sun_family);

if (bind(socket_unix, (struct sockaddr *)&local, len) == -1) {
    perror("bind");
    exit(1);
}

system("ls -l /tmp/");

unlink(local.sun_path);
return 0;
}

```

3.5. TCP/IP networking

Active le protocole TCP/IP.

3.5.1. IP: multicasting

Permet d'envoyer des paquets à plusieurs ordinateurs en même temps. Cette fonctionnalité est, par exemple, utilisée pour la diffusion audio et vidéo.

3.5.2. IP: advanced router

Par défaut, la décision du routage se fait en examinant l'adresse de destination. En activant cette option, on peut contrôler beaucoup plus précisément le routage et la prise de décision pourra se faire en fonction de nombreux autres critères.

Sous-option de advanced router

policy routing

Permet le remplacement de la table de routage classique, basée sur les adresses de destination, par la Base de Données des Politiques de Routage ou *Routing Policy DataBase* (RPDB) en anglais. Cette base de données est une liste ordonnée de règles qui scrutent certains caractéristiques des paquets :

- adresse source
- adresse de destination
- champ TOS
- marque du packet
- interface d'entrée

Si un paquet satisfait les spécifications d'une règle, alors l'action correspondante est réalisée. L'action standard consiste à fournir l'adresse IP du prochain saut. Si vous souhaitez plus d'informations sur ce sujet, je vous conseille la lecture de deux documents :

- La documentation ip-cref d'Alexey Kuznetsov, disponible avec le paquet `iproute`.
- L'article *Policy Routing for Fun and Profit*⁴.
- L'article *Policy Routing in Linux*⁵.

De plus, un exemple complet de routage avancé peut être consulté dans le document *LARTC : bases de données des politiques de routage*⁶.

Si l'option IP: use netfilter MARK value as routing key est validée, le routage des paquets pourra s'établir en fonction de la marque du paquet. Un exemple peut être consulté dans le document *LARTC : Netfilter et iproute - marquage de paquets*⁷.

Si l'option IP: fast network translation address est validée, le routeur pourra modifier les adresses source et destination des paquets transmis.

Exemple 3. Exemple de NAT

Soit un routeur avec d'un côté un réseau local 192.168.1.0/24 et de l'autre un réseau public (200.200.200.0/24, par exemple) ayant une connectivité sur Internet. On souhaite qu'une machine du réseau local (192.168.1.1, par exemple) soit reconnue avec l'adresse 200.200.200.10 sur Internet.

⁴ <http://www.linuxjournal.com/article/7134>

⁵ http://www.inetdoc.net/pdf/Policy_Routing_in_Linux_ENG.pdf

⁶ <http://www.inetdoc.net/guides/lartc/lartc.rpdb.html>

⁷ <http://www.inetdoc.net/guides/lartc/lartc.netfilter.html>

```
$ ip route add nat 200.200.200.10 via 192.168.1.1
$ ip rule add prio 300 from 192.168.1.1 nat 200.200.200.10
```

equal cost multipath

Avec cette option, on peut spécifier plusieurs routes alternatives que peuvent emprunter les paquets. Le routeur considère toutes ces routes comme étant de coûts égaux et choisit l'une d'elle d'une manière non déterministe si un paquet arrive avec la bonne correspondance.

Exemple 4. Exemple de chemins multiples

Considérons un routeur avec deux liaisons PPP. On souhaite que les paquets sortant puissent utiliser indifféremment ppp0 ou ppp1 comme interface de route par défaut.

```
$ ip route add default scope global nexthop dev ppp0 nexthop dev ppp1
```

use TOS value as routing key

L'entête d'un paquet IP contient un champ de 8 bits nommé *Type Of Service* (Type de service). Dans ce champ, il y a trois indicateurs qui permettent de préciser le type d'acheminement souhaité : Délai faible (faible temps d'attente), débit important et fiabilité importante. Cela permet de choisir entre, par exemple, une liaison satellite à haut débit mais avec un délai d'attente important ou une ligne louée à faible débit et faible délai. Cette option permet d'utiliser la valeur du champ TOS dans la liste de règles.

Exemple 5. Exemple d'utilisation du champ TOS pour le routage

But : tous les paquets marqués avec le champ TOS «débit important» (0x08) (par exemple le transfert de données via FTP) doivent emprunter une liaison RNIS.

```
$ip rule add tos 0x08 prio 100 table 10
$ip route add default dev ippp0 table 10
```

verbose route monitoring

Permet l'affichage de messages au sujet du routage.

large routing tables

Si la table de routage possède plus de 64 entrées, il est préférable d'activer cette option pour accélérer le processus de routage.

3.5.3. IP: kernel level autoconfiguration

Cette option permet de configurer les adresses IP des périphériques au moment du démarrage, ainsi que la table de routage. Les informations nécessaires à cette configuration sont fournies soit sur la ligne de commande, soit par l'intermédiaire des protocoles DHCP, BOOTP ou RARP.

Les informations sont fournies au noyau via le paramètre *ip*. Cette option est principalement utilisée pour la mise en place de stations clientes sans disque dur et qui ont besoin de monter la racine du système de fichiers via NFS. Pour plus d'informations, voir le fichier `Documentation/nfsroot.txt` dans les sources du noyau.

Exemple 6. Exemple de configuration IP au démarrage

```
LILO: linux ip=192.168.1.1::192.168.1.254:255.255.255.0:Linuxbox:eth0:none
```

3.5.4. IP: optimize as router not host

Permet de supprimer certaines vérifications lorsque le noyau reçoit un paquet. Dans le cas où Linux est principalement utilisé comme un routeur, c'est-à-dire une machine qui ne fait que transmettre les paquets, cela permet d'améliorer la vitesse de commutation.

3.5.5. IP: tunneling

Le *tunneling* permet l'encapsulation d'un protocole réseau dans un autre protocole réseau. Cette option permet l'encapsulation du protocole IP dans IP. Cela peut être utilisé dans le cas où l'on souhaite pouvoir faire communiquer deux réseaux ayant des adresses privées, donc non routables, à travers l'Internet. Un exemple complet de tunnel IP dans IP pourra être consulté dans le document *LARTC : IP dans un tunnel IP*⁸.

3.5.6. IP: GRE tunnel over IP

GRE est un protocole de tunnel qui a été originellement développé par CISCO™, et qui peut réaliser plus de choses que le tunnel IP dans IP. Par exemple, on peut aussi transporter du trafic multi-diffusion et de l'IPv6 à travers un tunnel GRE. Un exemple complet de tunnel GRE pourra être consulté dans le document *LARTC : Le tunnel GRE*⁹.

3.5.7. IP: TCP Explicit Congestion Notification support

Fonctionnalité qui permet aux routeurs d'annoncer aux clients une congestion du réseau.

3.5.8. IP: TCP syncookie support

Prévient une attaque appelée le *SYN Flooding*.

⁸ <http://www.inetdoc.net/guides/lartc/lartc.tunnel.ip-ip.html>

⁹ <http://www.inetdoc.net/guides/lartc/lartc.tunnel.gre.html>

3.5.9. IP: Allow large windows (not recommended if <16 Mb of memory)

Permet de définir de plus gros tampons dans lesquels les données sont stockées avant d'être envoyées à l'hôte destinataire.

3.5.10. Network packet filtering (replace ipchains)

Cette option active la fonction de filtrage des paquets traversant la machine Linux. Le filtrage permet un blocage sélectif du trafic IP en fonction, par exemple, de l'origine ou de la destination.

Sous-option de Network packet filtering

Network packet filtering debugging

Permet d'avoir des messages supplémentaires du code netfilter.

Bridge IP/ARP packet filtering

Permet d'utiliser les fonctionnalités de filtrage netfilter lorsque la machine est configurée pour fonctionner comme un pont.

3.5.10.1. IP: Netfilter configuration

Permet de rentrer dans un nouveau menu pour la configuration du filtrage. Celui-ci permet l'ajout de fonctionnalités dont voici une liste des plus importante :

Sous-option de Netfilter configuration

Connection tracking (required for masq/NAT)

Cette option permet de mettre en place le filtrage dit *StateFul*. Cette technique permet de garder en mémoire, dans une table d'état, une trace des «communications en cours». Cela permet de différencier le trafic entre les hôtes pairs, en émission et en réception. Cette option est indispensable pour l'utilisation des mécanismes de traduction d'adresse.

Connection mark tracking support

Cette option permet d'activer le support du marquage des communications. Ce support est nécessaire pour le critère de sélection connmark (voir [connection mark match support](#)) et pour la cible CONNMARK (voir [Packet Mangling](#)).

IP tables support (required for filtering/masq/NAT)

Cette option permet la mise en place de la structure générale pour le filtrage, le masquage ou la traduction d'adresse des paquets.

limit match support

Permet de limiter le débit en fonction d'une règle de correspondance.

Exemple 7. Limitation des requêtes ICMP

But : limiter les requêtes «pings» à 1 paquet par seconde.

```
# iptables -A INPUT -p icmp -icmp-type echo-request -m limit --limit 1/second -j ACCEPT
```

IP range match support

Permet de spécifier un intervalle d'adresses source ou destination.

Exemple 8. consultation d'un intervalle d'adresses IP source

But : Accepter de transmettre tous les paquets dont l'adresse source est comprise entre 192.168.1.1 et 192.168.1.10.

```
# iptables -A FORWARD -m iprange --src-range 192.168.1.1-192.168.1.10 -j ACCEPT
```

Exemple 9. consultation d'un intervalle d'adresses IP destination

But : Accepter de transmettre tous les paquets dont l'adresse destination est comprise entre 192.168.1.1 et 192.168.1.10.

```
# iptables -A FORWARD -m iprange --dst-range 192.168.1.1-192.168.1.10 -j ACCEPT
```

MAC address match support

Permet de baser le filtrage sur les adresses MAC des trames Ethernet.

Exemple 10. Filtrage de paquets sur l'adresse MAC

But : interdire les paquets provenant d'une adresse MAC particulière.

```
# iptables -A INPUT -m mac --mac-source 00:A0:24:A0:A4:11 -j DROP
```

Packet type match support

Permet de considérer le type de paquet : unicast, broadcast ou multicast.

Exemple 11. Journalisation des paquets de diffusion

```
# iptables -A INPUT -m pkttype --pkt-type broadcast -j LOG
```

netfilter MARK match support

Permet de baser le filtrage sur la marque d'un paquet. Le marquage d'un paquet est réalisé grâce à la cible MARK.

Exemple 12. Filtrage de paquets suivant le marquage

But : interdire les paquets à destination du serveur web local en utilisant le marquage de paquets.

```
# iptables -A PREROUTING -t mangle -p tcp --dport 80 -j MARK --set-mark=2
# iptables -A INPUT -m mark --mark 2 -j DROP
```

multiple port match support

Permet de spécifier un ensemble de ports sources ou destinations TCP ou UDP.

Exemple 13. Filtrage de paquets sur plusieurs ports source

But : interdire les accès aux ports sources 3000 et 4000.

```
# iptables -A INPUT -p tcp -m multiport --source-ports 3000,4000 -j DROP
```

TOS match support

Permet de baser le filtrage sur la valeur du champ TOS du paquet.

Exemple 14. Exemple de filtrage basé sur la valeur du champ TOS

But : marquer les paquets qui ont le champ TOS *Minimize-Delay* activé. Ce marquage pourra ainsi être utilisé pour le routage de ces paquets.

```
# iptables -t mangle -A PREROUTING -m tos --tos Minimize-Delay -j MARK --set-mark=1
```

recent match support

Permet de baser le filtrage en recherchant la présence d'une adresse dans une liste.

Exemple 15. Exemple de filtrage basé la présence d'une adresse dans une liste

But : FIXME

```
# iptables -A FORWARD -m recent --rcheck --seconds 60 -j DROP
# iptables -A FORWARD -i eth0 -d 127.0.0.0/8 -m recent --set -j DROP
```

ECN match support

La RFC3168 définit un mécanisme de notification de congestion. Ce mécanisme utilise les bits 7 et 8 du champs Type Of Service et l'en-tête IPv4 et définit deux nouveaux flags dans l'en-tête TCP. Ces flags ont pour noms CWR pour Congestion window Reduced et ECE pour ECN-Echo.

Exemple 16. Exemple de filtrage basé

But : FIXME

```
# iptables
```

DSCP match support

Permet de baser le filtrage sur la valeur des 6 bits DSCP de l'en-tête IP.

Exemple 17. Exemple de filtrage basé

Placer dans la classe Diffserv EF les paquets à destination de serveurs web.

```
# iptables -t mangle -A FORWARD -p tcp --dport 80 -j DSCP --set-dscp-class EF
```

```
Header Length: 20 bytes
  Differentiated Services Field: 0xb8 (DSCP 0x2e: Expedited Forwarding; ECN: 0x00)
    1011 10.. = Differentiated Services Codepoint: Expedited Forwarding (0x2e)
    .... ..0. = ECN-Capable Transport (ECT): 0
    .... ..0 = ECN-CE: 0
Total Length: 60
Identification: 0xfe4a (65098)
```

AH/ESP match support

FIXME

Exemple 18. Exemple de filtrage basé

But : FIXME

```
# iptables
```

LENGTH match support

Permet de baser le filtrage sur la longueur, exprimée en octets, du paquet IP.

Exemple 19. Exemple de filtrage basé sur la longueur de paquet

But : Supprimer les paquets ICMP de type *echo-request* qui ont une longueur supérieure à 84 octets.

```
# iptables -A FORWARD -p icmp --icmp-type echo-request -m length --length ! :84 -j DROP
```

TTL match support

Permet de baser le filtrage sur la valeur du champ TTL *Time To Live* de l'en-tête du paquet IP.

Exemple 20. Exemple de filtrage basé sur le champ TTL

But : FIXME

```
# iptables
```

tcpmss match support

Permet de baser le filtrage sur la valeur de l'option MSS *Maximum Segment Size* du protocole de transport TCP.

Exemple 21. Exemple de filtrage basé sur l'option MSS

But : Autoriser l'émission de paquet TCP dont le MSS est inférieur à 1400.

```
# iptables -A OUTPUT -o ppp0 -p tcp -m tcpmss --mss 0:1400 -j ACCEPT
```

Helper match support

Permet de baser FIXME

Connection state match support

Permet de baser le filtrage sur l'état des «communications».

Exemple 22. Filtrage de paquets FIXME

But : interdire les nouvelles communications entrantes vers la machine locale, mais autoriser les communications depuis la machine locale.

```
# iptables -P INPUT DROP
# iptables -A INPUT -i eth0 -m state --state NEW,INVALID -j DROP
# iptables -A INPUT -i eth0 -m state --state ESTABLISHED,RELATED -j ACCEPT
```

Connection tracking match support

Permet d'activer le module de critère de sélection conntrack. Cette option permet une plus grande granularité de recherche dans le suivi de communication.

Exemple 23. Exemple de filtrage de paquets en utilisant le suivi de communication avancé

But : autoriser les paquets appartenant à une connexion TCP établie.

```
# iptables -A FORWARD -m conntrack --ctstate ESTABLISHED,RELATED --ctproto tcp -j ACCEPT
```

Owner match support

Permet de baser le filtrage sur l'identifiant du processus local ayant créé le paquet. Cette option ne peut être utilisée que dans la chaîne OUTPUT.

Physdev match support

Dans le cas où la machine est configurée comme un pont entre deux réseaux Ethernet, cette option permet de repérer l'interface physique sur laquelle arrive les paquets ou sur laquelle ils doivent partir.

Exemple 24. Exemple de filtrage basé sur l'interface physique de sortie des paquets

But : Enregistrer les paquets sortant par l'interface eth0 d'un pont ethernet.

```
# iptables -A FORWARD -m physdev --physdev-out eth0 -j LOG --log-level 7
```

address type match support

Permet de baser le filtrage en fonction de la nature de l'adresse.

Exemple 25. Exemple de filtrage basé sur le type de paquet

But : accepter les paquets multicast.

```
# iptables -A INPUT -m addrtype --dst-type MULTICAST -j ACCEPT
```

realm match support

Permet de baser FIXME

SCTP protocol match support

Permet de baser FIXME

comment match support

Permet de baser FIXME

connection mark match support

Permet d'activer le critère de recherche connmark. Ce critère permet de repérer la marque associée à une communication.

Exemple 26. Limitation des requêtes ICMP

But : Autoriser les paquets sortant par eth0 et appartenant à la communication repérée par la marque 1.

```
# iptables -A OUTPUT -o eth0 -m connmark --mark 1 -j ACCEPT
```

hashlimit match support

Permet de baser FIXME

Packet filtering

Cette option permet le support du filtrage de paquets. La table gérant le filtrage se nomme *filter* et possède les chaînes par défaut INPUT, FORWARD et OUTPUT. La sous option *REJECT target support* ajoutera la cible REJECT. Cette cible permet de renvoyer un paquet d'erreur à la machine émettrice, simulant ainsi l'absence d'un service.

Exemple 27. Emission d'un paquet tcp-reject

But : rejeter les connexion ssh provenant des machines du réseau 192.168.0.0 et émettre un paquet tcp-reset.

```
# iptables -A INPUT -p tcp -s 192.168.0.0/24 --dport 22 \
-j REJECT --reject-with tcp-reset
```

LOG target support

Cette option permet l'enregistrement des paquets grâce au démon `syslogd`.

Exemple 28. Journalisation de paquets web

But : enregistrer tous les paquets à destination d'un site web sortant d'un hôte.

```
# iptables -A OUTPUT -p tcp --dport 80 \
-j LOG --log-level 7 --log-prefix "Paquets WEB :"
```

ULOG target support

Permet d'activer la cible ULOG utilisée pour envoyer à travers une socket netlink le paquet à un processus de l'espace utilisateur à des fins d'enregistrement.

TCPMSS target support

Permet de modifier la valeur du champ MSS contenu dans l'entête TCP grâce à la cible TCPMSS. L'usage le plus courant de cette règle de correspondance vise à adapter la taille maximale des segments TCP à la taille maximale des unités transmises au niveau réseau : MTU ou *Maximum Transmit Unit*.

Exemple 29. Exemple de filtrage basé sur l'option MSS

But : Adapter la taille maximale des segments TCP en fonction de la taille maximale des unités transmises au niveau réseau.

```
# iptables -A POSTROUTING -o ppp0 -p tcp -m tcpmss --mss 1400:1536 \
-j TCPMSS --clamp-mss-to-pmtu
```

Full NAT

Cette option permet le support de la traduction des adresses source (SNAT) et destination (DNAT).

La sous option MASQUERADE target support permet l'utilisation du *masquerading* comme traducteur d'adresse source.

La sous option REDIRECT target support permet la redirection des paquets vers la machine locale. Cette cible est utilisée dans la mise en place de proxys transparent.

La sous option NETMAP target support permet d'activer la cible NETMAP. Cette cible permet de rediriger le trafic destiné aux hôtes d'un réseau vers les hôtes d'un autre réseau.

La sous option SAME target support permet d'activer la cible SAME. Cette cible permet de traiter le cas particulier d'une traduction d'adresse source où plusieurs adresses de traduction peuvent être utilisées. Avec cette cible, on s'assure que la même adresse source sera utilisée pour tous les paquets d'une même communication.

Exemple 30. Exemple de traduction d'adresse source

But : masquer l'adresse source des paquets sortant par l'interface eth0 avec l'adresse 200.200.200.1.

```
# iptables -t nat -A POSTROUTING -o eth0 -j SNAT --to 200.200.200.1
```

Exemple 31. Exemple de traduction d'adresse de destination

But : rediriger les paquets à destination d'un serveur web vers le proxy de la machine 200.200.200.1 qui écoute le port 8080.

```
# iptables -t nat -A PREROUTING -p tcp --dport 80 -j DNAT --to 200.200.200.1:8080
```

Packet Mangling

Cette option permet de modifier certains éléments du paquet.

La sous option TOS target support permet de modifier le champ TOS du paquet.

La sous option ECN target support permet de supprimer les bits ECN.

La sous option MARK target support permet de marquer les paquets. Ces marques pourront être utilisées par la suite pour, par exemple, imposer un routage particulier ou par l'outil de configuration de la qualité de service `tc`.

La sous option CLASSIFY target support permet de positionner des paquets dans des classes de qualité de service.

La sous option TTL target support permet de modifier la valeur du champ TTL dans l'entête IP.

La sous option CONNMARK target support permet de marquer les paquets appartenant à une communication. Ces marques pourront être utilisées par la suite pour, par exemple, imposer un routage particulier.

Exemple 32. Configuration du champ TOS

But : positionner le champ TOS à *Minimize-Delay* pour les clients telnet.

```
# iptables -A PREROUTING -t mangle -p tcp --dport telnet -j TOS --set-tos Minimize-Delay
```

Exemple 33. Utilisation de la cible TTL

But : rendre invisible le routeur aux *traceroute*.

```
# iptables -A PREROUTING -p tcp --dport 33434:33542 -j TTL --ttl-inc 1
```

Exemple 34. Utilisation des cibles MARK et CONNMARK

But : positionner une marque pour tous les paquets d'une connexion.

```
# iptables -A POSTROUTING -t mangle -j CONNMARK --restore-mark ❶
# iptables -A POSTROUTING -t mangle -m mark ! --mark 0 -j ACCEPT ❷
# iptables -A POSTROUTING -t mangle -p tcp --dport 21 -j MARK --set-mark 1 ❸
# iptables -A POSTROUTING -t mangle -j CONNMARK --save-mark ❹
```

- ❶ Permet d'imposer aux paquets appartenant à une connexion la marque définie pour le premier paquet de cette connexion.
- ❷ Si le paquet possède déjà une marque, celui-ci est accepté car cela signifie que ce n'est pas le premier paquet d'une connexion.
- ❸ Le premier paquet d'une nouvelle connexion FTP est marqué avec la marque 1.
- ❹ La marque du premier paquet sera considérée comme étant la marque de tous les paquets de la connexion. Cette marque sera imposée aux autres paquets associés à la connexion grâce à la première règle.

3.6. 802.1Q VLAN Support

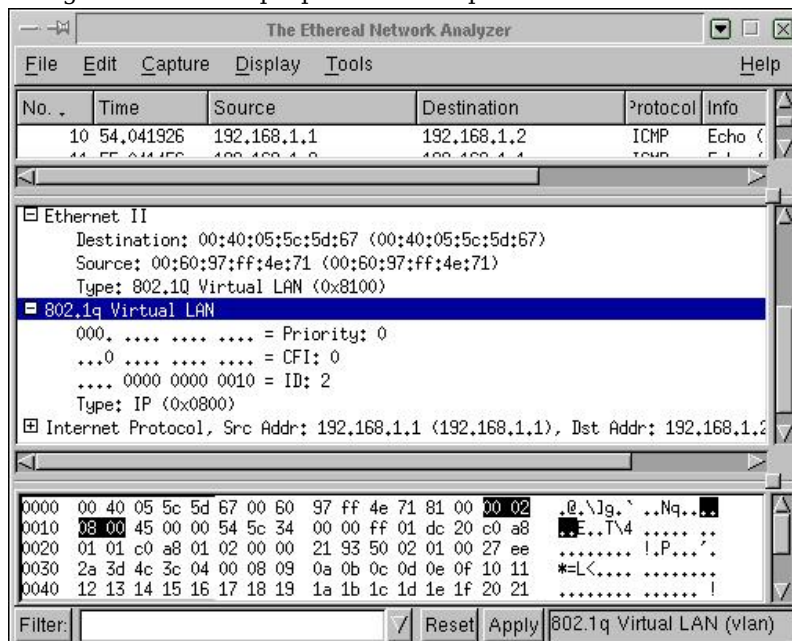
De nos jours, les réseaux locaux ou LANs (*Local Area Network*) sont définis comme étant un domaine de diffusion unique. Les réseaux locaux virtuels ou VLANs permettent de définir des réseaux locaux logiques sans se soucier de la localisation physique du matériel. Ces VLANs sont identifiés par une balise. Suivant la configuration du commutateur, il est parfois nécessaire d'indiquer à quel VLAN appartient une trame qui est envoyée sur le réseau. C'est ici que le protocole IEEE 802.1Q intervient. Ce protocole va permettre le marquage des trames Ethernet.

Exemple 35. Exemple de configuration du support VLAN

But : définir l'interface eth0 comme appartenant au VLAN numéro 2.

```
LinuxBox# modprobe 8021q
LinuxBox# ifconfig eth0 0.0.0.0 up
LinuxBox# vconfig add eth0 2
LinuxBox# ifconfig eth0.2 192.168.1.1 up
```

La figure ci-dessous propose un exemple de trame Ethernet avec le support du protocole IEEE 802.1Q.



Le document *Routage Inter-VLAN*¹⁰ développe beaucoup plus en détails l'utilisation des VLANs sur GNU/Linux.

¹⁰ <http://www.inetdoc.net/articles/inter-vlan-routing/>

3.7. The IPX Protocol

Ajoute le support du protocole IPX, principalement utilisé par les réseaux Novell™. IPX est un protocole de couche 3 (couche réseau).

3.8. Appletalk DDP

Appletalk est le protocole de communication des ordinateurs Apple™. Cette option permet à la machine Linux de pouvoir dialoguer avec les machines Apple™. En utilisant le programme netatalk, Linux peut agir comme serveur d'impression et de fichiers pour Mac. Il pourra également accéder aux imprimantes Appletalk.

3.9. DECnet support

La société Digital Equipment Corporation™ a créé une architecture réseau complète qui porte le nom de DNA (*Digital Network Architecture*). Cette architecture s'appuie sur une pile de protocole qui englobe l'ensemble des couches du modèle OSI. Les produits qui implémentent l'architecture DNA sont appelés produits DECnet. Par abus de langage, on désigne parfois le terme DECnet pour identifier un réseau DNA.

Pour plus d'informations sur ce sujet, consulter le site officiel du support *DECnet for Linux*¹¹.

3.10. 802.1d Ethernet Bridging

Avec cette option, votre boîte Linux pourra être assimilée à un pont Ethernet ; ce qui signifie que les différents segments Ethernet connectés apparaîtront comme un seul réseau Ethernet. Plusieurs ponts de ce type peuvent fonctionner ensemble pour créer un réseau de segments Ethernet encore plus grand en utilisant l'algorithme de *Spanning Tree Protocol* (IEEE 802.1d). Comme le protocole IEEE 802.1d est un standard universel, les ponts Linux fonctionneront correctement avec des équipements tiers.

Pour utiliser un pont Ethernet, vous aurez besoin des outils de configuration de pont. Voir *Documentation/networking/bridge.txt* pour plus d'information. Lire aussi le *Linux BRIDGE-STP-HOWTO*¹².

Notez que si votre machine fonctionne en pont, elle contient plusieurs interfaces Ethernet. Le noyau n'est pas capable de reconnaître plus d'une interface au démarrage sans assistance. Pour plus de détails, lire le document *Technologie Ethernet*¹³.

4. Les outils réseaux du noyau Linux

Le code des outils de configuration réseau ne faisant pas partie du noyau est généralement appelé : *userspace code*.

4.1. Configuration des interfaces réseaux

À partir de la version 2.2 du noyau LINUX, de nombreuses fonctionnalités sont apparues dans le support du protocole TCP/IP, notamment au niveau du routage. Pour pouvoir utiliser ces nouveautés, les outils classiques tels que **ifconfig** ou **route** ne suffisent plus. Il convient d'utiliser un nouvel outil, appelé **iproute2**. Le paquet de la distribution *Debian GNU/Linux* est baptisé **iproute**.

La syntaxe générale pour l'outil **iproute2** est la suivante :

```
Usage: ip [ OPTIONS ] OBJET { COMMAND | help }
où OBJET := { link | addr | route | rule | neigh | tunnel |
             maddr | mroute | monitor }
OPTIONS := { -V[ersion] | -s[tatistics] | -r[esolve] |
             -f[amily] { inet | inet6 | dnet | link } | -o[neline] }
```

Les différents objets permettent de voir ou de configurer un élément du réseau.

4.2. ip link

L'objet **link** permet de visualiser l'état des périphériques réseaux et de les modifier. La syntaxe générale pour cette option est la suivante :

```
Usage: ip link set DEVICE { up | down | arp { on | off } |
                        dynamic { on | off } |
                        multicast { on | off } | txqueuelen PACKETS |
                        name NEWNAME |
                        address LLADDR | broadcast LLADDR |
                        mtu MTU }
ip link show [ DEVICE ]
```

Cette option ne s'intéresse qu'au niveau 2 du modèle OSI.

4.3. ip address

Cet objet permet d'attacher une ou plusieurs adresses IPv4 ou IPv6 à un périphérique réseau.

¹¹ <http://sourceforge.net/apps/mediawiki/linux-decnet/>

¹² <http://en.tldp.org/HOWTO/BRIDGE-STP-HOWTO/>

¹³ <http://www.inetdoc.net/articles/ethernet/>

```
Usage: ip addr {add|del} IFADDR dev STRING
       ip addr {show|flush} [ dev STRING ] [ scope SCOPE-ID ]
                               [ to PREFIX ] [ FLAG-LIST ] [ label PATTERN ]
IFADDR := PREFIX | ADDR peer PREFIX
         [ broadcast ADDR ] [ anycast ADDR ]
         [ label STRING ] [ scope SCOPE-ID ]
SCOPE-ID := [ host | link | global | NUMBER ]
FLAG-LIST := [ FLAG-LIST ] FLAG
FLAG := [ permanent | dynamic | secondary | primary |
         tentative | deprecated ]
```

La configuration de base d'une interface réseau ressemble à ceci :

```
# ifconfig eth0 192.168.0.2 netmask 255.255.255.0 broadcast 192.168.0.255
# route add default gw 192.168.0.1
```

Les commandes équivalentes avec l'outil `iproute2` sont les suivantes :

```
# ip addr add 192.168.0.2/24 dev eth0 broadcast 192.168.0.255
# ip route add default dev eth0 via 192.168.0.1
```

4.4. ip rule

Le routage du trafic IP a été complètement revu avec le noyau 2.2. Avant cette version, la prise de décision ne se faisait qu'en consultant l'adresse de destination. Dans certaines circonstances, on peut souhaiter router les paquets IP en se basant sur d'autres champs : adresse source, champs TOS, etc.

Le routage est maintenant basé sur l'existence d'un ensemble de règles, qui dirigent le paquet vers des tables de routage. L'ensemble de ces règles est vu comme une base de données par le noyau, que l'on appelle *Routing Policy DataBase* (RPDB). Cette base de données de la politique de routage est en fait une liste linéaire de règles ordonnées par une valeur numérique de priorité. La gestion de ces règles se fait par l'intermédiaire de l'objet `rule`, dont voici la syntaxe :

```
Usage: ip rule [ list | add | del ] SELECTOR ACTION
SELECTOR := [ from PREFIX ] [ to PREFIX ] [ tos TOS ] [ fwmark FWMARK ]
           [ dev STRING ] [ pref NUMBER ]
ACTION := [ table TABLE_ID ] [ nat ADDRESS ]
          [ prohibit | reject | unreachable ]
          [ realms [SRCREALM/]DSTREALM ]
TABLE_ID := [ local | main | default | NUMBER ]
```

Chaque règle est constituée d'un sélecteur et d'une action. Quand le noyau a besoin de prendre une décision sur le routage, la *Routing Policy DataBase* (RPDB) est scannée dans l'ordre des priorités croissantes. Pour chaque paquet, on compare le sélecteur de la règle et l'en-tête du paquet. Si il y a correspondance entre les deux, l'action est réalisée. En général, l'action consiste à se «brancher» sur une table de routage qui contient l'information utile. Si l'action ne parvient pas à déterminer une route, alors la règle suivante est examinée. La commande suivante permet de lister l'ensemble des règles définies dans la base RPDB :

```
$ ip rule ls
0: from all lookup local
32766: from all lookup main
32767: from all lookup default
```

Ces lignes méritent quelques explications. Les chiffres de la colonne de gauche indiquent la priorité de la règle. Ensuite, on a le sélecteur. Dans ce cas, toutes les règles seront appliquées à tous les paquets (*from all*). Enfin, on a l'action. Le mot-clé `lookup` indique d'aller regarder la table de routage dont le nom suit.

4.5. ip route

Une fois que le noyau a sélectionné la table à consulter, il recherche dans celle-ci les informations de routage proprement dites. Ces informations précisent le périphérique de sortie et éventuellement l'adresse de la prochaine passerelle. Par défaut, il y a trois tables de routage : `local`, `main` et `default`.

1. `local` : cette table est une table un peu spéciale ayant la plus grande priorité. Elle contient les routes pour les adresses locales et les adresses de diffusion.
2. `main` : cette table est la table de routage normale, et ce sont les informations contenues dans celle-ci qui seront affichées par la commande `ip route ls`.
3. `default` : cette table est généralement vide et n'est consultée que si les règles précédentes n'ont pas sélectionné le paquet.

La syntaxe associée à l'objet `route` est la suivante :

```
Usage: ip route { list | flush } SELECTOR
       ip route get ADDRESS [ from ADDRESS iif STRING ]
                               [ oif STRING ] [ tos TOS ]
       ip route { add | del | change | append | replace | monitor } ROUTE
SELECTOR := [ root PREFIX ] [ match PREFIX ] [ exact PREFIX ]
           [ table TABLE_ID ] [ proto RTPROTO ]
           [ type TYPE ] [ scope SCOPE ]
ROUTE := NODE_SPEC [ INFO_SPEC ]
NODE_SPEC := [ TYPE ] PREFIX [ tos TOS ]
```

```

        [ table TABLE_ID ] [ proto RTPROTO ]
        [ scope SCOPE ] [ metric METRIC ]
INFO_SPEC := NH OPTIONS FLAGS [ nexthop NH ]...
NH := [ via ADDRESS ] [ dev STRING ] [ weight NUMBER ] NHFLAGS
OPTIONS := FLAGS [ mtu NUMBER ] [ advmss NUMBER ]
        [ rtt NUMBER ] [ rttvar NUMBER ]
        [ window NUMBER ] [ cwnd NUMBER ] [ ssthresh REALM ]
        [ realms REALM ]
TYPE := [ unicast | local | broadcast | multicast | throw |
        unreachable | prohibit | blackhole | nat ]
TABLE_ID := [ local | main | default | all | NUMBER ]
SCOPE := [ host | link | global | NUMBER ]
FLAGS := [ equalize ]
NHFLAGS := [ onlink | pervasive ]
RTPROTO := [ kernel | boot | static | NUMBER ]

```

5. Configuration du filtrage

5.1. Introduction

La sécurité informatique est un terme général qui cache de nombreux aspects, tels que la sécurité physique de la machine, le contrôle d'accès aux fichiers, etc. L'un des aspects de la sécurité concerne la sécurité des réseaux. Avec la démocratisation d'Internet, les tentatives d'intrusions se développent. Afin de limiter le nombre de ces attaques, le mieux est encore de filtrer dès l'entrée du réseau tout ce qui n'est pas censé y entrer. Le système qui permet la mise en place de ce filtrage s'appelle un *firewall* ou un «pare-feu» en français.

Un *firewall* peut se définir comme un dispositif de protection (matériel et/ou logiciel) constituant un filtre entre un ordinateur ou un réseau local et un réseau non sûr (Internet ou un autre réseau local par exemple). On distingue deux grandes familles de *firewalls* :

- Les *firewalls* basés sur le filtrage réseau. Ces éléments fonctionnent au niveau *transmission de l'information* des couches du modèle OSI. Le filtrage s'effectue en fonction des informations contenues dans les en-têtes des trames, des paquets (adresses source et destination) et des segments (ports source et destination). Ce type de filtrage ne s'intéresse pas au contenu des paquets.
- Les *firewalls* applicatifs ou de services. Ce type de filtrage permet de contrôler le *traitement de l'information*. Dans ce cas, l'information contenue dans le paquet peut être prise en compte. Les demandes de connexions sont dirigées vers un programme spécial appelé mandataire ou proxy de service. C'est ce dernier qui établira la connexion vers le service extérieur demandé.

5.2. Netfilter et iptables

Pour pouvoir bénéficier des fonctions de filtrage réseau du noyau LINUX, il faut y intégrer l'option Network packet filtering lors de la compilation. Cette fonctionnalité est une structure générale qui permet à d'autres éléments de se «brancher» dessus. Pour pouvoir indiquer les différentes règles au noyau, on dispose de l'utilitaire appelé **iptables**.

L'outil **iptables** utilise le concept de tables de règles, chaque table correspondant à une fonctionnalité d'examen du paquet. La table *filter* correspond au filtrage des paquets, la table *nat* concerne la traduction d'adresse et la table *manget* permet la modification des paquets.

5.3. Utilisation de l'outil iptables

Dans un premier temps, **iptables** servira à la gestion des chaînes. Une chaîne peut être assimilée à une politique de sécurité associée à un flux de données. Par exemple, on peut définir une chaîne *INTERNET* pour désigner tous les flux venant de l'extérieur de votre réseau local. Trois chaînes par défaut existent, à savoir *INPUT*, *FORWARD* et *OUTPUT*. Si le nombre de règles est limité, on peut se contenter de celles-ci, mais si les règles deviennent conséquentes, il est préférable, pour faciliter la gestion, de créer de nouvelles chaînes. Les commandes de l'outil **iptables** associées à la gestion des chaînes sont les suivantes :

- N
Création d'une nouvelle chaîne. Exemple **iptables -N INTERNET**.
- X
Suppression d'une chaîne vide. Exemple **iptables -X INTERNET**.
- P
Mise en place de la règle par défaut pour une chaîne existante. Exemple : **iptables -P INPUT DROP**. Seules les chaînes *INPUT*, *FORWARD* et *OUTPUT* peuvent avoir une règle par défaut et les seules cibles disponibles sont *ACCEPT* et *DROP*.
- L
Lister les règles d'une chaîne. Exemple : **iptables -L INTERNET**.
- F
Effacer les règles d'une chaîne. Exemple : **iptables -F INTERNET**.

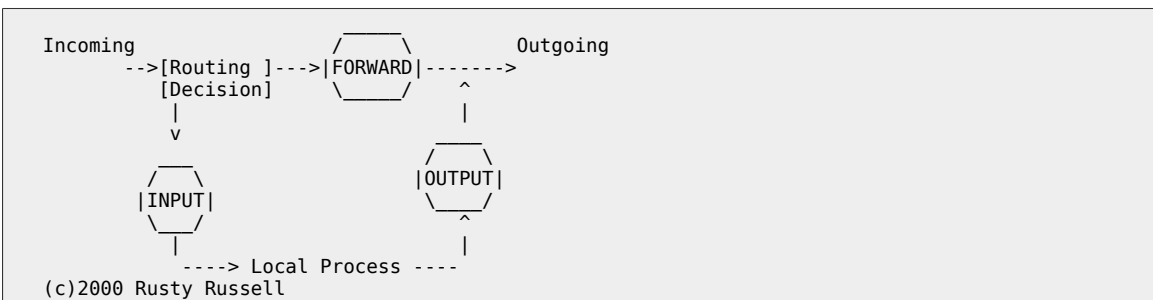
Dans un deuxième temps, il convient de construire les règles à l'intérieur des différentes chaînes. L'ajout d'une règle s'effectue avec l'option `-A` de l'outil **iptables**, tandis que l'effacement d'une règle se fait avec l'option `-D`. Les principales spécifications sur lesquelles les règles peuvent s'appuyer sont les suivantes :

- `-s` : spécifie l'adresse IP source
- `-d` : spécifie l'adresse IP de destination
- `-p` : spécifie le protocole. Le protocole peut être `tcp`, `udp` ou `icmp`
- `-i` : spécifie le nom de l'interface physique à travers laquelle les paquets entrent
- `-o` : spécifie le nom de l'interface physique à travers laquelle les paquets sortent

Ces spécifications sont les plus générales, mais il en existe bien d'autres qui sont parfaitement listées dans la page de manuel de l'outil **iptables**.

5.4. Le filtrage avec iptables

Avec **iptables**, les différentes règles de filtrage sont organisées et regroupées dans des chaînes. Par défaut, il y a trois chaînes appelées *INPUT*, *OUTPUT* et *FORWARD*. L'arrangement de ces chaînes est proposé sur le schéma suivant :



Les différentes chaînes sont consultées suivant la procédure suivante :

1. Quand un paquet arrive, le noyau décide de la destination de ce paquet : c'est la phase de routage.
2. Si le paquet est destiné à la machine, le paquet descend dans le diagramme et la chaîne *INPUT* est appliquée. Si le paquet passe cette chaîne, celui-ci sera transmis à l'un des processus locaux.
3. Si le routage décide que le paquet est destiné à un autre réseau, alors c'est la chaîne *FORWARD* qui est appliquée.
4. Enfin, les paquets envoyés par un processus local seront examinés par la chaîne *OUTPUT*. Si le paquet est accepté, celui-ci sera envoyé quelle que soit son interface de sortie.

Une chaîne est composée d'une liste de règles. Une règle décide de l'avenir d'un paquet en fonction de son en-tête. Les règles d'une chaîne sont examinées les unes après les autres jusqu'à ce qu'une correspondance soit trouvée. Finalement, si aucune correspondance n'est trouvée, la règle par défaut, *policy*, est appliquée. On associe à chaque règle une action à réaliser qui décide de l'avenir du paquet. Les fonctions principales sont les suivantes :

- **ACCEPT** : cette cible permet d'accepter les paquets.
- **DROP** : cette cible permet de refuser les paquets sans avertir le demandeur que sa demande de connexion a été refusée.
- **REJECT** : cette cible permet de refuser les paquets, mais en avertissant le demandeur que sa demande de connexion a été refusée en lui envoyant un paquet **RESET (RST)**.

5.5. Le suivi des communications, *stateful firewalling*

L'une des grandes nouveautés de la partie réseau du noyau 2.4 est la possibilité du suivi des communications. Ceci fait référence à la capacité du noyau à maintenir une table de suivi des communications en se basant, par exemple, sur le couple adresses (source et destination), sur les numéros de ports (source et destination), sur les types de protocoles ou l'état de la communication. Les pare-feux disposant de cette fonctionnalité sont appelés *stateful firewalls*. Dans ce cas, les paquets sont inspectés dans le contexte d'une session. Par exemple, un segment **TCP** avec le bit **ACK** activé sera rejeté si aucun segment **SYN** correspondant n'a été reçu auparavant.

Le suivi des «communications» se base sur trois états :

- **NEW** : correspond à la demande de communication **TCP** initiale, au premier datagramme **UDP** ou au premier message **ICMP**.

- **ESTABLISHED** : si une entrée de la table de suivi des communications correspond, alors le paquet appartient à une communication de type ESTABLISHED. Dans le cas du protocole TCP, on se réfère au bit ACK après qu'une communication ait été initiée. Dans le cas de datagrammes UDP c'est l'échange entre deux hôtes et les correspondances de numéros de ports qui sont prises en compte. Enfin, les messages ICMP echo-reply doivent correspondre aux requêtes echo-request.
- **RELATED** : se réfère aux messages d'erreurs ICMP correspondant à une «communication» TCP ou UDP déjà présente dans la table de suivi.

D'un point de vue pratique, le module de suivi des communications sera activé grâce à l'option `-m state` de la commande **iptables**. L'option `--state` permet de spécifier l'état de la communication à considérer.

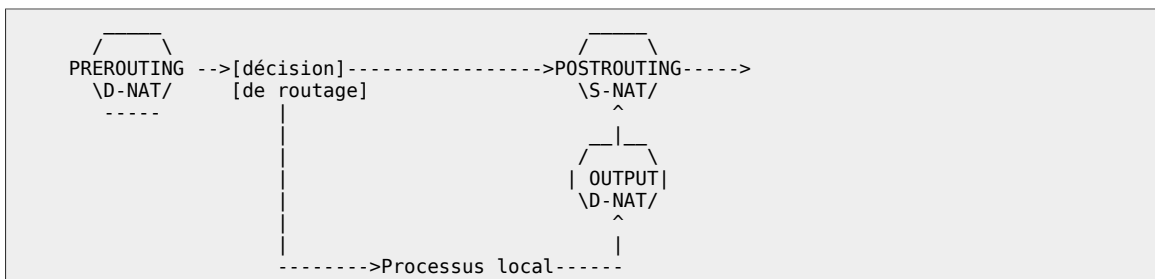
Exemple 36. Autorisation d'une connexion ssh vers l'extérieur

```
# iptables -A INPUT -m state --state ESTABLISHED,RELATED -j ACCEPT
# iptables -A OUTPUT -p tcp -d 0/0 --dport 22 -m state --state NEW -j ACCEPT
```

5.6. La traduction d'adresse (NAT)

La traduction d'adresse est une technique qui permet de remplacer une adresse source ou destination par une autre. La traduction d'adresse du noyau 2.4 supporte le *source NAT* (SNAT) et le *destination NAT* (DNAT). La table `nat` permet la modification des adresses source et destination grâce à deux chaînes par défaut :

- **PREROUTING** : permet la modification de l'adresse de destination (DNAT) avant que le paquet ne passe par les fonctions de routage.
- **POSTROUTING** : permet la modification de l'adresse source (SNAT) après que le paquet soit passé par les fonctions de routage.



Intéressons nous dans un premier temps à la traduction d'adresse source ou S-NAT. Il existe en deux formes distinctes au sein des noyaux (2.4|2.6) : SNAT et MASQUERADE. SNAT est la forme standard de la traduction d'adresse source, tandis que la deuxième est plus spécialisée au cas d'adresses IP assignées dynamiquement. La distinction entre les deux formes est subtile.

Avec SNAT, la communication est maintenue pendant un certain temps d'attente lors d'un dysfonctionnement. Si cette communication est rétablie suffisamment rapidement, les programmes réseaux ne seront pas affectés et le trafic TCP interrompu sera retransmis, dans la mesure où l'adresse IP n'a pas été changée.

Avec la forme MASQUERADE, il n'y a pas de temps d'attente quand la connexion est rompue et les informations concernant la traduction d'adresse sont effacées. Ceci permet d'utiliser immédiatement la nouvelle adresse IP qui peut être attribuée lors d'une reconnexion à un fournisseur d'accès, par exemple.

Exemple 37. SNAT standard

```
# iptables -t nat -A POSTROUTING -o eth0 -j SNAT --to-source 192.192.192.192
```

Exemple 38. MASQUERADE

```
# iptables -t nat -A POSTROUTING -o ppp0 -j MASQUERADE
```